UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE COMPARACIÓN DE PERFILES EN REDES SOCIALES

Juan Francisco Campo García
Tutor: Álvaro Ortigosa Juárez

Julio 2020

DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE COMPARACIÓN DE PERFILES EN REDES SOCIALES

AUTOR: Juan Francisco Campo García
TUTOR: Álvaro Ortigosa Juárez

Dpto. Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Julio de 2020

Resumen (castellano)

Este Trabajo Fin de Grado plantea el diseño y la implementación de un sistema de clasificación de perfiles. Dicho sistema tiene como objetivo la obtención de una lista de perfiles similares para un perfil dado de una conocida red social. Este sistema se valdrá de técnicas de procesamiento del lenguaje natural para el análisis de los perfiles, obteniendo características semánticas y estilísticas de ellos, y será desarrollado en Python. Durante el desarrollo de este TFG se explicarán en detalle los pasos seguidos para el diseño del sistema: la obtención de datos, el almacenamiento de éstos, los modelos de procesamiento del lenguaje natural utilizados, los resultados obtenidos para cada modelo y el análisis de todos los resultados.

Abstract (English)

This Bachelor Thesis proposes how to design and implement a profile classification system. This system pursues getting a list of profiles which are similar to a target profile from a known social network. The system will use Natural Language Processing techniques to analyse the profiles, retrieving semantic and stylistic properties from them, and it will be developed using Python. Throughout the development of this Bachelor Thesis the steps followed for the design of the system will be explained in detail. These steps are: retrieval of data, storing the data obtained, Natural Language Processing models used, results obtained for every model used and result analysis.

Palabras clave (castellano)

Clasificación, NLP, Doc2Vec, Perfiles, Redes Sociales, Similitud, Bag-of-words, Contexto, Vector, Párrafo

Keywords (inglés)

Classification, NLP, Doc2Vec, Profiles, Social Networks, Similarity, Bag-of-Words, Context, Vector, Paragraph

Agradecimientos

Este TFG no habría sido posible sin el apoyo de todas las personas que me han acompañado durante toda mi trayectoria y a las que quiero agradecer todo lo que han hecho por mí.

Agradezco a mis padres que hayan estado conmigo, tanto en los buenos momentos, en los que hemos disfrutado, como en los malos, en los que hemos llorado. Gracias por darme las oportunidades que tenido y allanarme el camino para llegar a ser quién soy.

Agradezco a Ana, mi hermana, toda la fuerza que me da como pequeño terremoto. Todo lo que hemos vivido juntos nos ha hecho más fuertes y sabios, pero sobre todo nos ha aportado una conexión especial. Sin ti me habría perdido muchas experiencias y no habría sido tan valiente como para estar hoy aquí.

A mis amigos les doy las gracias por haber decidido serlo. Habéis creído en mí incluso cuando yo no lo hacía y me habéis ayudado todo lo que os he dejado y más. Gracias a Jimena, a Souad, a Angélica, a Rubén, a David, a Juan, a Lorena, a Arturo, a Álex C., a Álex L., a Ana, a Irene, a Ángel, a Nick, a Eli, a Zaida, a Manuel, a Jorge, a Agus… Podría seguir y llenar la página porque para mí sois muy importantes y siempre os tengo presentes. De verdad, os aprecio muchísimo y valéis vuestro peso en oro y más.

También quiero dar las gracias a Idoia, por los consejos tan sabios que sabes dar y que me han ayudado una barbaridad. Sin ella, yo no estaría escribiendo este documento ahora mismo.

Álvaro, te agradezco toda tu paciencia y el haber sido un ángel caído del cielo. Gracias por ser mi tutor de proyecto.

Y, por último, quería dejar este espacio para agradecerle a Alberto el haber entrado en mi vida. Gracias por ser tú, por estar ahí, por preocuparte, por animarme, por compartir y por creer en mí. Eres mi luz entre las sombras y significas tantísimo para mí. Gracias de corazón.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

1 Introducción

1.1 Motivación

En las últimas décadas, con el auge de Internet, la vida diaria se desplaza con más frecuencia a este entorno, viendo como tareas que antes tenían lugar *in-situ* (por ejemplo, pagar recibos bancarios o hacer la compra) ahora se suceden en el espacio virtual. Esto también ocurre con los delitos, que han pasado a tener su lugar en el ciberespacio en forma de *ciberdelitos*, tales como acoso a personas vulnerables, incitación al odio de determinados grupos mediante la utilización del discurso del odio, amenazas, calumnias, etc.
Uno de los grandes focos donde tienen lugar este tipo de delitos son las Redes Sociales: espacios en los que, de forma anónima, se pueden realizar acciones sin exponerse públicamente y de forma cómoda, utilizando dispositivos electrónicos. Así pues, aumentan los casos en los que se hace necesario investigar perfiles de dichas redes y determinar en qué círculos se mueven dichas personas. Los analistas investigadores son los que desempeñan esta labor de investigación, en la que suele estar implicada la búsqueda de perfiles similares e intentar dilucidar si existe una conexión entre la persona a cargo del perfil de la búsqueda y los perfiles similares. En el caso de que existiera la conexión, se utilizarían estos perfiles para extraer más información de la persona.
En este contexto, es de utilidad una herramienta que, para un perfil anónimo, sea capaz de encontrar perfiles de identidad conocida, gestionados por el mismo usuario. Esta herramienta se ha de tener en cuenta como un apoyo, no como un sustituto a la labor del analista; es decir, la herramienta no trata de identificar no precisión un perfil sino más bien generar una lista de los perfiles que muestren más similitud, delegando la responsabilidad de la decisión de si los perfiles merecen ser objeto de estudio al analista.

1.2 Objetivos

Este TFG propone el diseño e implementación de un sistema informático que sea capaz de:

Dado un perfil de una red social, obtener una lista de perfiles que tengan características similares al introducido por orden de semejanza.

Se parte de la suposición de que la forma de escribir de cada persona es una característica individual e irreproducible y que por lo tanto sería posible la clasificación automática de perfiles o textos basándose en el análisis del lenguaje utilizado.
Para ello, se utilizarán técnicas de Procesamiento del Lenguaje Natural – NLP – para extraer las características de los perfiles y determinar las similitudes entre los distintos perfiles alojados en el sistema.
Las técnicas aplicadas en este proyecto se podrían utilizar para comparar perfiles en o entre distintas redes sociales, pero este trabajo se centrará exclusivamente en perfiles extraídos de la red social Twitter, puesto que es la plataforma que más cantidad de información abierta ofrece.

1.3 Organización de la memoria

La memoria consta de los siguientes capítulos:

- **1. Introducción**. 1.1. Motivación. 1.2. Objetivos 1.3. Organización de la memoria.
- **2. Estado del arte.** 2.1. Avances e investigaciones actuales. 2.2. Herramientas.
- **3. Pasos previos.** 3.1. Requisitos previos. 3.2. Obtención de muestras. 3.3. Tratamiento de muestras.
- **4. Procesamiento del lenguaje natural.** 4.1 Modelos utilizados.
- **5. Pruebas y análisis de resultados.** 5.1. Resultados obtenidos. 5.2. Análisis de resultados.
- **6. Conclusiones y trabajo futuro.** 6.1. Conclusiones. 6.2. Trabajo futuro.

2 Estado del arte

Dentro de los grandes avances que se están produciendo en el campo de la inteligencia artificial, se observa que un de área que está cobrando mucha importancia recientemente es la del Procesamiento del Lenguaje Natural (abreviado como NLP – *Natural Language Processing* –). El NLP se puede definir brevemente como la representación, análisis y generación de lenguaje humano o natural a través de la computación. Esta área se ha visto beneficiada tanto por los avances producidos en Inteligencia Artificial como por la disponibilidad de grandes cantidades de texto online (que sigue en aumento constante) y la tendencia creciente de utilizar los sistemas informáticos para interacciones (como por ejemplo a través de redes sociales).
Sus aplicaciones son variadas: desde su uso en la traducción automática hasta la aplicación en sistemas de respuesta a preguntas de toda índole.

2.1 Avances e investigaciones actuales

Recientemente hay varias líneas de investigación abiertas en el tema del procesamiento del lenguaje natural y varios avances realizados.

En el área de la búsqueda y análisis de texto se están dando muchos avances en poco tiempo. BERT es uno de estos avances: *Bidirectional Encoder Representations from Transformers.* [1] Es un modelo de representación del lenguaje presentado por Google que ha sido implementado en su motor de búsqueda y que se caracteriza por la utilización de redes neuronales. BERT está diseñado para hacer un pre-entrenamiento no supervisado de representaciones bidireccionales, utilizando datos sin etiquetar, mediante la consideración conjunta de los contextos del lado izquierdo y derecho de una palabra. Aunque el concepto es simple, en la práctica dota de una mejor capacidad de análisis de los textos y brinda un sinfín de oportunidades en áreas como la generación de texto.

Los sistemas de diálogo también están en auge en estos últimos años y se están volviendo esenciales en nuestra vida. Cada vez son más fáciles de usar y responden con mayor fluidez, consecuencia de los avances de la inteligencia artificial y del procesamiento del lenguaje natural. En los últimos años, han aparecido sistemas de diálogo orientados tanto a tareas generales como especializadas (como pueden ser los *chatbots* que aparecen en web médicas, para hacer consultas de sintomatología, etc). Si bien los primeros *chatbots* como ELISA estaban basados en reglas, en los últimos tiempos son más usuales los basados en datos, que se apoyan en la recuperación de información, en técnicas de *machine learning* o en ambas [2].

La generación del lenguaje natural es una de las tareas más generales en el procesamiento del lenguaje natural y está presente por ejemplo en los sistemas de diálogo anteriores. A pesar de que los modelos son mejores con el tiempo, esto no conseguía que la generación de lenguaje fuera mejor y dejase de producir artefactos. Líneas de investigación recientes consistentes en mejorar las técnicas de muestreo, como el *nucleus sampling* [3] o nuevas funciones de pérdida [4], producen mejoras en los resultados obtenidos. La conclusión a la que llegan estas investigaciones es que para mejorar los resultados no sólo vale con

mejorar los modelos o las técnicas de búsqueda, sino que hay que mejorar ambas cuestiones a la par para que haya mejorías.

Los modelos actuales de procesamiento del lenguaje natural están basados predominantemente en redes neuronales profundas que son opacas en términos de cómo llegan a ciertas predicciones. Esta limitación ha llevado a un creciente interés en la creación de modelos más interpretables que puedan revelar el razonamiento subyacente a ciertas salidas de los modelos. ERASER (*Evaluating Rationales And Simple English Reasoning*) es un *benchmark* creado para ayudar en la investigación de modelos del procesamiento del lenguaje natural que sean más interpretables que los actuales. [5]

En cuanto a traducción automática, en los últimos años se ha visto que las herramientas de traducción, antes utilizadas como meros diccionarios para recuperar los significados de los términos puesto que la fiabilidad a la hora de traducir textos completos era bastante baja, son capaces de entender el contexto general de las frases y realizar traducciones que, si bien necesitan revisión, están muy próximas a las que podría realizar una persona con un conocimiento extenso en las lenguas origen y destino.

A continuación, se exponen algunas herramientas para desarrollos en temas de procesamiento del lenguaje natural:

2.2 Herramientas

2.2.1 NLTK

Natural Language Toolkit – o NLTK – [6] es un set de herramientas codificadas en Python para la realización de tareas del campo del procesamiento del lenguaje natural tales como clasificación de textos, obtención de los lexemas de las palabras, o análisis del lenguaje. Esta herramienta trae consigo una colección de más de cincuenta corpus y recursos léxicos para la realización de pruebas y desarrollos. Gracias a esta colección de recursos, NLTK es un buen punto de partida para comenzar en el procesamiento del lenguaje natura, incluso si no se dispone de conocimientos específicos en el tema.

2.2.2 SpaCy

SpaCy [7] es una librería *open-source* dedicada al procesamiento del lenguaje natural. Destaca en el análisis sintáctico de textos, que resulta muy útil a la hora de interpretar y clasificar sentimientos presentes en los textos, así como para la optimización de interfaces conversacionales. También destaca a la hora de localizar y clasificar entidades en los textos en categorías predefinidas (nombres de personas, organizaciones, lugares, etc.).

2.2.3 AllenNLP

AllenNLP [8] es una herramienta desarrollada sobre las librerías de PyTorch, utilizadas en el campo del aprendizaje automático. Utiliza las librerías de SpaCy para el preprocesamiento de datos, mientras que el resto de las tareas las realiza utilizando sus propios recursos.
AllenNLP cuenta con una serie de modelos que son utilizados como recursos para la creación de interfaces conversacionales de mayor complejidad, como es el caso de los modelos de vinculación textual, que garantizan la generación de textos coherentes y comprensibles para las personas, mejorando así la interacción entre usuarios y máquinas.
Esta herramienta se puede utilizar para la realización de tareas simple o complejas, teniendo un buen rendimiento en ambos casos.

2.2.4 GenSim

Gensim [9] es una herramienta utilizada para la extracción de información; más concretamente se trata de una librería de procesamiento del lenguaje natural *open-source*, diseñada para la exploración de documentos y el modelado de temas. La propiedad característica de Gensim es la utilización de vectores de palabras, representando los documentos como secuencias de vectores y clusters.
Los principales casos de uso de Gensim son el análisis de datos, la búsqueda semántica y la generación de texto.

2.2.5 TextBlob

TextBlob [10] es una herramienta para el procesamiento del lenguaje natural alimentada por NLTK. Se puede usar para el análisis de sentimientos, a través de interfaces conversacionales, o para traducción automática, utilizando su corpus de idiomas.

3 Pasos previos

Este TFG plantea la creación de un sistema que sea capaz de almacenar una serie de perfiles encontrados en una red social y que, al recibir un perfil, devuelva una lista de perfiles en el sistema que sean similares al de entrada.

3.1 Requisitos previos

El sistema a diseñar se va a integrar en otro sistema de mayor complejidad y con mayor alcance de lo que se tiene en cuenta en este proyecto. Por lo tanto, han de tenerse en cuenta ciertas directrices iniciales durante el diseño del sistema, como que se ha de desarrollar el código utilizando para ello el lenguaje de programación Python.

Así pues, todas las decisiones de diseño están influenciadas de base por este requisito no funcional y todas las herramientas utilizadas tendrán que ser compatibles con este lenguaje de programación, es decir, a lo largo de este TFG sólo aparecerán expuestas herramientas que sean compatibles con Python.

Teniendo presente lo anunciado anteriormente, se procede a la primera etapa del proyecto.

3.2 Obtención de muestras

En primer lugar, es necesario recopilar muestras de perfiles reales con los que poder trabajar de cara a etapas más avanzadas del proyecto. Además, dado que el sistema se va a utilizar en entorno en producción, las pruebas con datos reales aportan un plus de significancia al estudio.

La plataforma elegida desde la cuál se obtendrán los perfiles con los que se trabajará es la red social Twitter.
El siguiente paso es determinar cómo descargar los datos necesarios y posibilidades de automatizar este proceso.

Twitter cuenta con una API propia que, a través de peticiones, permite realizar consultas y obtener datos diversos para su descarga (como puedan ser los seguidores de un perfil, los tweets publicados, etc.). No obstante, la API tiene limitaciones, entre ellas que no es posible recuperar todos los tweets publicados por un perfil desde el inicio de los tiempos, sino que sólo deja recuperar los últimos 3200 tweets publicados desde la fecha de la consulta.

Esta cantidad de tweets podría ser suficiente para entrenar, pero en perfiles con mayor volumen de tweets, una muestra tan pequeña podría inducir a error y no dar una representación temática y singular adecuada del perfil, ya que se busca caracterizar el perfil tanto por ideas del autor como por el estilo empleado. Por esta razón se decide incorporar una herramienta que permita obtener, sino todos los tweets asociados, casi todos los tweets de un perfil.

En este contexto se decide utilizar Twint: una herramienta desarrollada en Python para el *scraping* de tweets, pudiendo hacerlo anónimamente, sin necesidad de autenticarse siquiera en la plataforma.
La herramienta tiene multitud parámetros de configuración, permitiendo hacer tareas variadas, como por ejemplo, obtener todos los tweets de un usuario que fueran publicados antes de una fecha, o después de una fecha dada; obtener todos los tweets que contengan un término de búsqueda proporcionado; si se desea obtener la localización del usuario cuando se publicaron los tweets o el cliente de Twitter desde el que fue publicado el tweet, etc. También permite discernir y recuperar únicamente tweets con contenido multimedia, como fotos o videos. Así mismo, permite almacenar los tweets recuperados en distintos formatos, como JSON, CSV o SQLite.
A pesar de todo, la recuperación de tweets es lenta y una descarga del orden del millón de tweets puede tardar fácilmente un día completo.

En este proyecto, se decide descargar tweets de cuarenta y tres perfiles distintos, con diferentes temáticas.

Ciberseguridad	Ordenadores	Cocina	Videojuegos	Arquitectura
CnecEs, ICFS_UAM, EEI_ICFS, CursosDeForense, sandra_vazquezb, C1b3rWall, pedaguitu, ivanPorMor, gIntelSeg, AngelGdeAgreda, AgdaMundo, JDiaz_Caneja, Inteligencia_L, selvaorejon, onBRANDING, avalos_morer, CosasDeHackers, TheXXLMAN, INCIBE, CCNCERT, CybercampEs, EdgarVasquezC	LasCosasRetro, uto_dev, hardware_espana, easysmartech, infocomputer	karguinano, pesadillacocina, rosaarda, canalcocina, cocinaTelefe	EspVideojuegos, VideojuegosGAME, IGN_es, Capcom_Es, blissy, Jose_Strife, NintendoEs,	ETSAMadrid, Arsitek_Estudio, DesignCarmenG, funarco

Tabla 1. Perfiles de Twitter agrupados por temática.

Entre los perfiles con temática "*Ciberseguridad*" existen cuentas que son llevadas por la misma persona. Se quiere comprobar si es posible que, además de agrupar los perfiles por temática similar, el sistema sea capaz de determinar si los perfiles que son llevados por la misma persona tienen alto parecido entre ellos y es posible agruparlos.

Con la lista de perfiles a obtener determinada, se utiliza la herramienta Twint para la descarga. Los nombres de usuario son pasados uno a uno y se almacenan todos los tweets recuperados en ficheros, uno por usuario, en formato JSON. Este almacenamiento es

intermedio, puesto que el sistema final se conectará a una base de datos de la que recuperará los tweets.


```
{"id": 1259597293419999232, "conversation_id": "1259451028040306691", "created_at": 1589146371000, "date": "2020-05-10", "time":
{"id": 1259040906302406659, "conversation_id": "1259040906302406659", "created_at": 1589013718000, "date": "2020-05-09", "time":
{"id": 1258733240078217216, "conversation_id": "1258455271946534913", "created_at": 1588940365000, "date": "2020-05-08", "time":
{"id": 1258721232645763072, "conversation_id": "1258455271946534913", "created_at": 1588937502000, "date": "2020-05-08", "time":
```


Figura 1. Ejemplo de fichero de almacenamiento de tweets en formato JSON.

Con los datos presentes en los ficheros, se procede a la siguiente etapa del proyecto.

3.3 Tratamiento de muestras

Tras haber completado la descarga de los datos desde la red, es preciso almacenarlos de forma que se pueda acceder al conjunto de los perfiles de forma rápida y que, además, se pueda ampliar el número de perfiles almacenados.
Dado que el sistema en el que se va a integrar el proyecto de este TFG consta de una base de datos relacional, gestionada mediante PostgreSQL, se decide usar este tipo de base de datos, para trabajar con un entorno lo más similar posible al de producción.

En este TFG se va a utilizar únicamente el contenido escrito de cada tweet para el desarrollo del proyecto, así pues, de los datos descargados previamente sólo son de interés los campos *id*, *username*, y *tweet*. El resto de los campos se obvian y no tienen cabida en este proyecto.
Por tanto, es necesario generar una tabla con tres columnas:

- Una que almacene el identificador único y funcione como clave primaria. Dado que los identificadores pueden alcanzar valores muy grandes, es preciso utilizar una variable de gran tamaño.
- Una columna que almacene el usuario que publicó el tweet.
- Una última columna que almacene el contenido del tweet, es decir, el texto escrito.

Las tareas de creación de tablas y población de las mismas son fácilmente automatizables; sólo es necesario utilizar una herramienta con la capacidad de conectar con la base de datos.
Para ello, se utilizará la librería Psycopg 2, un adaptador de las bases de datos gestionadas por PostgreSQL desarrollado para Python. Su funcionamiento es sencillo: se conecta a una base de datos especificada utilizando un usuario dado; después se obtiene un cursor, con el que se pueden realizar diferentes operaciones; se ejecutan las operaciones con el cursor, se obtiene los resultados, en caso de que los haya, y, si es necesario, se hace que los cambios realizados en la base de datos se hagan permanentes antes de cerrar la sesión.

Así pues, se genera un script que conecta con la base de datos y trata de crear la tabla en la que se almacenarán los tweets. Una vez creada la tabla, se procede a leer los ficheros de los perfiles en formato JSON y se almacenan los campos enunciados en párrafos anteriores, poblando la tabla.

El total de tweets introducidos en la base de datos es del orden de 790 000 tweets aproximadamente, lo cual se considera una muestra apta para el trabajo.

Una vez se tienen todos los datos de muestra descargados y debidamente almacenados, se procede al desarrollo del sistema de clasificación.

4 Procesamiento del lenguaje natural

El procesamiento del lenguaje natural – referido como NLP de ahora en adelante – es la parte fundamental de este proyecto. La idea es ser capaz de establecer las relaciones de similitud entre distintos perfiles utilizando las características sintácticas y semánticas para ello.

La herramienta que se va a utilizar para el NLP es Gensim, una librería capaz de analizar la estructura semántica de diversos textos y encontrar diferentes características, como los temas presentes en ellos o el estilo utilizado para la redacción.

Antes de continuar, se van a mencionar los conceptos de documento, corpus, vector y modelo a partir de este punto, por lo que se procede a su explicación:

- Un documento hace referencia a la unidad más pequeña con la que se trabaja que tiene sentido por sí sola. Cuando se habla de documentos se piensa en textos, como puede ser un libro o, en este proyecto, cada uno de los perfiles a utilizar. Se podría considerar que un tweet es un documento, pero dado que la intención es comparar perfiles completos, el tweet sería una unidad demasiada pequeña para poder llevar a cabo la tarea, teniendo que juntar una colección de tweets y compararlos contra otra colección de tweets.
- El corpus es una colección de documentos. Cada uno de los perfiles de Twitter obtenido se agrupa en esta colección y forma una nueva entidad, que es lo que se utilizará en el sistema para entrenar y obtener temas e ideas contenidos en el texto y otras características únicas.
- Un vector es la representación matemática de un documento. Es necesario que se conviertan las palabras contenidas en los documentos a una forma matemática, ya que el sistema trabaja con valores cuantitativos, y no cualitativos, como son las palabras.
- Un modelo no es más que un algoritmo de transformación de representación de los documentos, permitiendo cambiar de una representación inicial a otra. Al trabajar con vectores, lo que se hace es variar entre espacios vectoriales.

Con estos conceptos en mente, se procede a explicar los pasos seguidos durante el desarrollo del proyecto.
Se quiere conseguir que, al introducir un documento, el sistema lo procese y sea capaz de obtener los documentos similares al de entrada. Para ello el primer paso es formar los documentos que utilizará el sistema para comparar.
Esto se hace realizando una consulta a la base de datos y recuperando el contenido de cada uno de los tweets asociados a un usuario. El contenido de cada tweet se agrega a una variable, el documento, de tal forma que todos los tweets de un mismo usuario formen parte del mismo documento. Se repite el proceso para cada uno de los diferentes perfiles, resultando en un corpus.
Una vez se tiene el corpus, el siguiente paso es preprocesar el contenido de los documentos del corpus en vistas a generar una partición de entrenamiento. Se puede preprocesar el contenido de muchas formas, como puede ser eliminar las palabras vacías (palabras que no añaden ningún tipo de valor semántico) presentes en los documentos. En este proyecto se ha realizado un preprocesamiento simple, consistente en separar las palabras que aparecen

en el texto utilizando como separador el espacio (“ “), eliminar la distinción entre letras mayúsculas y minúsculas, y eliminar las palabras que tengan una longitud muy pequeña (menos de dos letras) o muy grande (más de quince letras).

Con el corpus preprocesado, se procede a seleccionar el modelo a utilizar para el entrenamiento.

4.1 Modelos utilizados

La herramienta cuenta con distintos modelos que han sido utilizados para la realización de pruebas, cada uno con sus propias características intrínsecas. A continuación, se explican uno por uno los modelos utilizados.

4.1.1 Term Frequency – Inverse Document Frequency

Este modelo utiliza un corpus en un formato concreto para la inicialización: el formato *bag-of-words*.
Bag-of-words [11] es un modelo de representación que consiste en representar textos como si de un set de palabras se tratase. Con este modelo no se tienen en cuenta características gramaticales del lenguaje ni el orden en el que aparecen las palabras en los textos, sino que sólo se conserva la multiplicidad, es decir, el número de veces que se registra cada palabra.

Para esta representación, se genera un diccionario que contiene todas las palabras que aparecen en los distintos documentos presentes en el corpus. Una vez se tiene el diccionario, se representa cada documento como un vector que contiene la frecuencia de aparición de cada palabra que aparece en el texto y está registrada en el diccionario.

Figura 2. Ejemplo de codificación bag-of-words.

Por ejemplo, se tiene un diccionario que contiene las palabras “are”, “cat”, “dog”, “is”, “now” ,“on”, “table” y “the”, en ese orden; por otro lado, tenemos la frase “the dog is on the table”, que se representaría con el vector [0, 0, 1, 1, 0, 1, 1, 1] (la palabra “are” no aparece en la frase, “cat” tampoco, “dog” aparece una vez, y así sucesivamente).

Tras recibir el corpus con representación *bag-of-words*, el modelo TF-IDF entrena con los vectores asociados a cada documento del corpus, buscando las palabras que menos se repiten en todo el corpus y asignando un mayor valor numérico a estas palabras frente a otras más comunes. Por tanto, este modelo da mayor importancia a términos con un ratio de aparición bajo frente a términos más frecuentes.
Los vectores devueltos por el modelo tienen la misma dimensionalidad que los vectores iniciales (es decir, si el diccionario contiene 50 000 palabras, los vectores tendrán una

dimensionalidad de [1, 50 000]), pero los valores de los vectores se ven afectados, pasando de vectores de valores enteros a vectores con valores reales.

Volviendo al fundamento teórico, este modelo utiliza un estadístico que refleja lo importante que es una palabra en un documento dentro de un corpus. Para ello, se utilizan los siguientes estadísticos: la frecuencia de término – *term frequency* – y la frecuencia inversa de documento – *inverse document frequency*. [12]

La frecuencia del término – *tf* – para cada documento se puede calcular de varias maneras, de las que se enuncian algunas:

- Binaria: la frecuencia es 1 si aparece en el documento, o 0 en caso contrario.
- Absoluta: la frecuencia es simplemente el número de veces que aparece el término en el documento.
- Ajustada: viene a ser el número de veces que aparece el término partido del número de palabras totales en el documento.
- Aumentada: el número de veces que aparece el término en el documento se divide por la frecuencia absoluta del término que más aparece en el documento.

La frecuencia inversa de documento – *idf* – es un estadístico global y también se puede calcular de distintas maneras, siendo la más habitual el logaritmo de: el número de documentos entre el número de documentos donde aparece el término.

Con estos estadísticos se obtiene el valor de la frecuencia de término – frecuencia inversa de documento mediante el producto de *tf* e *idf*.
Valores numéricos altos en este estadístico indican que el término aparece muchas veces en el documento, pero no es un término usual en el corpus; de esta forma se les resta valor a los términos comunes en los documentos (como puedan ser las palabras vacías) y se le da mayor importancia a palabras clave para la clasificación.

4.1.2 Latent Semantic Indexing

El modelo del que se habla a continuación es conocido como un método de recuperación de información cuya característica principal es la extracción de las ideas principales existentes en un corpus, estableciendo asociaciones entre los términos que aparecen en contextos similares.
Latent Semantic Indexing – LSI a partir de ahora – está basado en el principio de que los términos que se utilizan en las mismas situaciones suelen tener el mismo significado.
LSI se caracteriza por utilizar una técnica matemática de nombre Descomposición en valores singulares – *Singular Value Decomposition* o SVD – a través de la cuál se identifican patrones en las relaciones entre los términos y conceptos encontrados en una colección de textos. [13]

SVD es una factorización de una matriz, que dice que una matriz A se puede descomponer en otras tres matrices distintas: T, S y D.

$$A \approx TSD^T$$

Figura 3. Singular Value Decomposition.

Aplicado al modelo actual, la matriz A es del tipo término-documento, siendo las filas los términos que aparecen en el corpus y las columnas representan los documentos; cada celda contiene el número de veces que aparece el término en el documento. Una vez se ha construido la matriz, se pueden aplicar funciones de ponderación locales y globales a los datos. Esta matriz es similar a la presentada en el modelo TF-IDF y se construye utilizando también el modelo de representación *bag-of-words*.
La matriz T es una matriz de vectores término-concepto, que define el espacio vectorial de los términos; la matriz S es una matriz de valores singulares, que expresa la información conceptual derivada del corpus, y la matriz D es un matriz de vectores concepto-documento, que define el espacio vectorial de los documentos.

La SVD es truncada mediante un factor k para reducir la dimensionalidad de las matrices. Esto se hace para reducir el ruido en los datos y eliminar artefactos que estaban presentes en la matriz original A, conservando la información semántica de carácter importante presente en los textos. Uno de los problemas encontrados es determinar qué factor k es el óptimo para utilizar con SVD y obtener los mejores resultados. Estudios indican que un valor de aproximadamente 300 dimensiones es el que suele dar los mejores resultados para un corpus de alrededor de cientos de miles de documentos. [14]
No obstante, estudios recientes indican que un valor entre 50 y 1000 dimensiones da buenos resultados, en función de la naturaleza y el tamaño del corpus utilizado. [15]

A diferencia del modelo anterior, el modelo LSI permite añadir más documentos al modelo en cualquier momento. Esto es posible ya que se actualiza el modelo subyacente mediante cambios incrementales, proceso conocido como *online training*.

4.1.3 Random Projections

Random Projections [16] es un modelo construido a partir del modelo *Term Frequency – Inverse Document Frequency* que busca reducir la dimensionalidad del espacio vectorial.
Para ello, se vale de una matriz de valores aleatorios de dimensión d x k, siendo d el número de dimensiones de los datos originales y siendo k la nueva dimensionalidad de los datos.
Como se enuncia en el modelo *Latent Semantic Indexing*, reducir la dimensionalidad mantiene la información relevante a la vez que reduce el ruido presente en los datos, obteniendo mejores resultados. Este tipo de reducción tiene menor coste computacional que la asociada a otras técnicas, como la SVD.
No obstante, a pesar de aplicar la reducción, el modelo base sigue siendo TF-IDF, por lo que los procedimientos son exactamente los mismos, y se espera que los resultados obtenidos sean similares, aunque con mejoras.

4.1.4 Latent Dirichlet Allocation

El modelo *Latent Dirichlet Allocation* – de ahora en adelante LDA – [17] es un modelo estadístico de tipo generativo (basado en probabilidad conjunta), en contraposición con los modelos de tipo discriminativo (basados en probabilidad condicional).

Este modelo se utiliza para clasificar documentos automáticamente, valiéndose de los temas inferidos en el corpus utilizado.
Se considera que cada documento contiene en su interior una colección variada de temas, y el modelo le asigna una serie de temas predominantes a cada documento. LDA introduce una distribución a priori Dirichlet, que codifica la idea de que cualquier documento trata un reducido número de temas y que cada tema utiliza sólo un conjunto de palabras de forma frecuente. Es decir, si se tiene un tema como pueda ser "música", es probable que se encuentren palabras del estilo de "sonido", "canción", o "partitura" de forma frecuente.
Hay que destacar que ningún tema está definido de forma concisa, ni semántica ni epistemológicamente, por lo que una palabra puede estar asociada a varios temas, teniendo una probabilidad asociada diferente en función del tema.

En cuanto a fundamento matemático, se debe destacar que el modelo LDA es básicamente el modelo bayesiano del modelo *Probabilistic Latent Semantic Analysis* – pLSA, visto anteriormente como modelo LSI (notar que el modelo LSI y el modelo LSA son dos nomenclaturas distintas que se usan para referirse al mismo modelo). Se puede considerar que LDA es una generalización de pLSA.

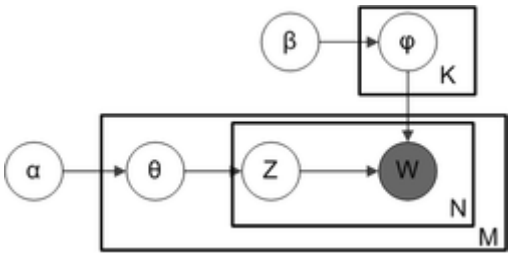

Figura 4. Modelo LDA.

El modelo consta de:

- El número de documentos, denotado por M.
- El número de palabras en un documento i, denotado por N_i
- El parámetro de Dirichlet para la distribución de temas por documento, denotado por α.
- El parámetro de Dirichlet para la distribución de palabras por tema, denotado por β.
- La distribución de temas en el documento i, denotada por θ_i.
- La distribución de palabras para el tema k, denotada por φ_i.
- El tema para la palabra j del documento i, denotado por z_{ij}.
- Una palabra determinada, denotada por w_{ij}.

4.1.5 Hierarchical Dirichlet Process

Hierarchical Dirichlet Process – en adelante, HDP – es un modelo bayesiano no paramétrico que se utiliza para modelar datos de diferentes orígenes y características, de forma no supervisada. Tiene un funcionamiento similar al encontrado en el modelo LDA, con la ventaja de que para una colección de documentos dada se infiere a posteriori el número de temas a utilizar y se caracteriza la distribución de los temas.

El modelo de temas no parámetrico considera que los documentos son grupos de palabras observadas, que los temas son distribuciones de términos y que cada documento presenta los temas con diferentes proporciones a las del resto.
El modelo HDP busca una estructura latente no dimensional que pueda ser usada para tareas como clasificación, exploración y síntesis. [18]

4.1.6 Doc2Vec

Se presenta a Doc2Vec como un modelo en el que cada documento es representado como un vector; destaca su capacidad para el aprendizaje no supervisado, trabajando con datos que no han sido etiquetados previamente. [19]

Los modelos basados en *Bag-of-words*, además de no conservar el orden de aparición de las palabras, no tratan de aprender el significado de éstas y, como consecuencia, la distancia entre vectores no refleja la diferencia en significado; es por eso que se utiliza el modelo Word2Vec para modelar los datos, modelo sobre el que está construido Doc2Vec pero con modificaciones.

Word2Vec es un modelo que integra palabras en un espacio vectorial de menor dimensión utilizando para ello una red neuronal poco profunda. Con esto se obtienen vectores asociados a palabras en los que los vectores que están cercanos en el espacio vectorial tienen significados similares, basándose en el contexto, y los vectores que están lejanos entres si tienen significados distintos. Por ejemplo, fuerte y poderoso estarían cercanos pero fuerte y papilla estarían lejanos. [20]

Dentro del modelo Word2Vec se observan dos modelos de representación:

- El modelo *Skip-gram*, que coge pares de palabras y entrena una red neuronal con una capa oculta con el objetivo de que para una palabra dada se obtenga la distribución de probabilidad de las palabras con las que se suele juntar para la construcción de frases.
- El modelo *Continuous-bag-of-words*, que es similar al modelo *Skip-gram*. Consta de una red neuronal con una capa oculta, pero difiere en que utiliza las palabras cercanas para obtener la palabra central.

Con el modelo Word2Vec es posible calcular los vectores para cada palabra que está presente en un documento, pero el objetivo es calcular vectores para un documento completo. Una aproximación sería utilizar todos los vectores de todas las palabras presentes en el documento y hacer una estimación con estos vectores del que sería el vector resultante, pero existe otro método mejor, que es el que implementa el modelo Doc2Vec: asociar a cada documento un vector propio que contribuye en el entrenamiento y es actualizado, al igual que los vectores de palabras.

Doc2Vec implementa el *Paragraph vector*, un framework no supervisado que aprende representaciones vectoriales con distribución continua para fragmentos de textos [21]. Los textos pueden tener longitud variable, siendo desde frases hasta documentos de texto completos.

La representación en vectores es entrenada para ser capaz de predecir palabras en un párrafo. Explicado con más detalle, la predicción se llevaría a cabo de la siguiente forma: se concatena el vector del párrafo con varios vectores de palabra provenientes del párrafo y

se predice la próxima palabra para un contexto dado. Tanto los vectores de palabra como los vectores de párrafo se entrenan mediante un gradiente estocástico descendiente y retropropagación.
Mientras que los vectores de párrafo son únicos, es decir, hay un vector por párrafo, los vectores de palabra se comparten entre todos los párrafos.

Cada palabra es representada con un vector que es concatenado o promediado con otros vectores de palabra en el contexto dado y el vector resultante es utilizado para predecir otras palabras en este contexto.
Una vez el modelo ha sido entrenado, los vectores de palabras se mapean en un espacio vectorial de tal forma que las palabras semánticamente similares tienen representaciones vectoriales similares.

El modelo *Paragraph Vector* es capaz de construir representaciones a partir de secuencias de entrada de longitud variable. A diferencia de otros enfoques, este es general y se puede aplicar a todo tipo de textos: frases, párrafos o documentos completos.

Existen dos implementaciones de este modelo:

Paragraph Vector – Distributed Memory, en el que cada párrafo se mapea a un vector único, representado en la Figura 5 como una columna en la matriz D, y cada palabra se mapea a un vector único de la misma forma, representado como una columna en la matriz W. Los vectores de párrafo y de palabras son promediados o concatenados para predecir la siguiente palabra en un contexto.

El identificador de párrafo puede ser considerado como otra palabra, actuando como una memoria que recoge lo que falta del contexto actual o el tema del párrafo. Los contextos de palabras tienen una longitud fija y se muestrean de una ventana deslizante sobre el párrafo. El vector de párrafo es compartido con todos los contextos de palabras que se generan del mismo párrafo, pero no se comparten entre párrafos. La matriz de vectores de palabra W sí que se comparte entre los párrafos.

Los vectores de párrafo y de palabra se entrenan utilizando un gradiente estocástico descendente y el gradiente se obtiene vía retropropagación. En cada etapa del gradiente estocástico descendente, se puede muestrear un contexto de palabras de longitud fija de un párrafo aleatorio y se computa el gradiente de error de la red de la Figura 5. Este gradiente se utiliza para actualizar los parámetros del modelo.

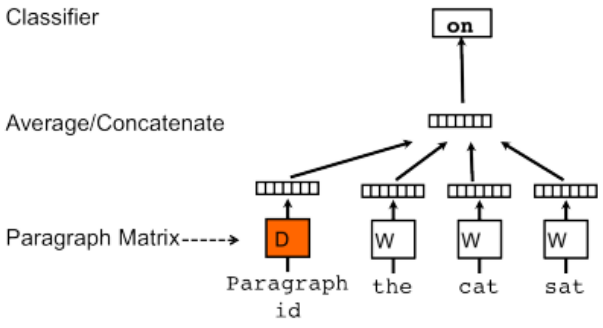

Figura 5. Paragraph Vector Model: Distributed Memory.

Como nota, decir el modelo *Distributed Memory* es el análogo del modelo *Continuous Bag-of-words* de Word2Vec.

Las ventajas de este modelo y los vectores de párrafo son que el aprendizaje se realiza a partir de datos sin etiquetar y trabajan bien en tareas en las que no se tienen suficientes datos etiquetados. Además, los vectores de párrafo superan algunas de las dificultades provenientes del modelo *bag-of-words*, como son la semántica de las palabras, ya que se aprende el significado a las palabras subyacentes en los párrafos; aparte hay que añadir que estos vectores consiguen guardar el orden de aparición de las palabras, por lo que conservan más información útil.

La otra implementación es *Paragraph Vector – Distributed Bag-of-Words*. Este método consiste en predecir palabras utilizando un párrafo, sin utilizar el contexto de las palabras presente en el párrafo; es decir, para un párrafo dado se busca qué posibles palabras se conseguirían sabiendo la naturaleza del párrafo.

En cada iteración del gradiente estocástico descendente se muestrea una ventana de texto, después se elige una palabra al azar de esta ventana de texto y se realiza una clasificación utilizando el vector de párrafo, como se ve en la Figura 6.

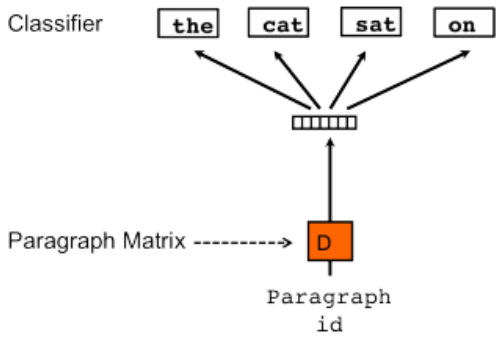

Figura 6. Paragraph Vector Model: Distributed Bag-of-words.

Además de ser conceptualmente simple, este modelo guarda menos datos que el modelo visto anteriormente: sólo se necesita almacenar los pesos softmax, en contraposición a los pesos softmax y los vectores de palabra que se usaban en el modelo *Distributed Memory*.

Este modelo es considerado el análogo de Word2Vec *Skip-Gram*.

Para el entrenamiento del modelo Doc2Vec se hace uso de los siguientes parámetros:

- El tamaño de vector de salida, utilizado para reducir la dimensionalidad del espacio vectorial.
- El mínimo número de apariciones de una palabra para ser tenida en cuenta en el entrenamiento y formar vectores de palabra con ella.
- El número de épocas que entrena el modelo. En el caso de utilizar corpus pequeños, se recomienda aumentar el número de épocas de entrenamiento, mientras que para corpus grandes se suele establecer un número entre 10 y 20 épocas.

Antes de comenzar el entrenamiento, es imprescindible que el modelo construya el vocabulario del corpus para tener registradas todas las palabras que aparecen y el recuento del número de apariciones de cada palabra.

5 Pruebas y análisis de resultados

Con los modelos enunciados en el capítulo 4 y los datos obtenidos en el capítulo 3 se procede a la obtención de resultados y a la valoración de los mismos.

5.1 Term Frequency – Inverse Document Frequency

Para este modelo se genera un corpus utilizando los 43 perfiles obtenidos, en formato Bag-of-words. Los perfiles son preprocesados de la siguiente forma: las palabras se separan por espacios y se cambian las mayúsculas por minúsculas.

Una vez se tiene el corpus formado, se pasa al modelo y se entrena. Para el modelo no es necesario utilizar ningún parámetro de entrenamiento distinto del corpus de entrada.

En la siguiente tabla se observan los resultados obtenidos para diez perfiles elegidos al azar, leídos de la siguiente manera: los diez perfiles elegidos al azar se representan en la columna de *Perfil introducido*, mientras que los dos perfiles que más similitud tienen con los perfiles elegidos se muestran en la columna de *Perfiles más similares*, siendo el valor que se encuentra entre paréntesis el porcentaje de similitud con el perfil introducido en tanto por uno.

Perfil introducido	***Perfiles más similares***
«agdamundo»	(0.1053): «pedaguitu» (0.0496): «avalos_morer»
«angelgdeagreda»	(0.0292): «agdamundo» (0.0232): «pedaguitu»
«arsitek_estudio»	(0.1106): «easysmartech» (0.0532): «funarco»
«avalos_morer»	(0.2917): «pedaguitu» (0.2515): «sandra_vazquezb»
«canalcocina»	(0.1221): «karguinano» (0.0666): «rosaarda»
«cneces»	(0.0814): «pedaguitu» (0.0756): «sandra_vazquezb»
«cosasdehackers»	(0.0260): «c1b3rwall» (0.0253): «avalos_morer»
«funarco»	(0.0945): «etsamadrid» (0.0532): «arsitek_estudio»
«gintelseg»	(0.6470): «ivanpormor» (0.1371): «inteligencia_l»
«nintendoes»	(0.0989): «espvideojuegos» (0.0663): «videojuegosgame»

Tabla 2. Term Frequency – Inverse Document Frequency.

Al estudiar los resultados obtenidos se observa que los valores de similitud son por lo general muy bajos, siendo el promedio de un 10% de coincidencia. Dejando de lado la

similitud, se observa que este modelo categoriza bien los perfiles por temática, quitando el caso del perfil *arsitek_estudio*, al que se empareja con *easysmartech*, cuando ambos perfiles pertenecen a categorías totalmente opuestas: el primero perteneciendo a *Arquitectura* y el segundo perteneciendo a *Ordenadores*. Sin embargo, a pesar de categorizar bien, no es capaz de establecer conexiones entre perfiles que son llevados por la misma persona; esto se observa en el perfil de *agdamundo*, entre cuyos perfiles más similares no aparece *angelgdeagreda*.

5.2 Latent Semantic Indexing

Se usa un corpus de *Bag-of-words* (el mismo que se utilizó para el modelo anterior), pero para este modelo se usa el parámetro *Topics*, que se utiliza para reducir la dimensionalidad.

A continuación, se recogen los datos para distintas pruebas en el que se varía el parámetro:

Perfil introducido	***Perfiles más similares***
«agdamundo»	(0.9968): «pedaguitu» (0.9951): «avalos_morer»
«angelgdeagreda»	(0.9562): «pesadillacocina» (0.9492): «selvaorejon»
«arsitek_estudio»	(0.9984): «infocomputer» (0.9918): «espvideojuegos»
«avalos_morer»	(0.9994): «pedaguitu» (0.9951): «agdamundo»
«canalcocina»	(0.9888): «karguinano» (0.9593): «icfs_uam»
«cneces»	(0.9981): «cybercampes» (0.9981): «c1b3rwall»
«cosasdehackers»	(0.9952): «cybercampes» (0.9948): «cneces»
«funarco»	(0.9979): «edgarvasquezc» (0.9973): «ccncert»
«gintelseg»	(0.9984): «c1b3rwall» (0.9981): «ccncert»
«nintendoes»	(0.9927): «funarco» (0.9911): «edgarvasquezc»

Tabla 3. Latent Semantic Indexing. Topics 5.

Perfil introducido	***Perfiles más similares***
«agdamundo»	(0.9811): «sandra_vazquezb» (0.9790): «pedaguitu»
«angelgdeagreda»	(0.5670): «arsitek_estudio» (0.5370): «uto_dev»
«arsitek_estudio»	(0.9279): «easysmartech» (0.8982): «selvaorejon»

«avalos_morer»	(0.9893): «pedaguitu» (0.9844): «sandra_vazquezb»
«canalcocina»	(0.8644): «capcom_es» (0.8536): «c1b3rwall»
«cneces»	(0.9737): «gintelseg» (0.9680): «cybercampes»
«cosasdehackers»	(0.9747): «sandra_vazquezb» (0.9670): «capcom_es»
«funarco»	(0.9901): «gintelseg» (0.9781): «edgarvasquezc»
«gintelseg»	(0.9944): «c1b3rwall» (0.9901): «funarco»
«nintendoes»	(0.9206): «cocinatelefe» (0.9153): «funarco»

Tabla 4. Latent Semantic Indexing. Topics 20.

En comparación con los resultados obtenidos para el modelo TF-IDF, se observa que la precisión a la hora de encontrar similitudes es mayor, pasando del 10% de coincidencia medio a rondar el 90% de coincidencia. No obstante, a pesar de que vemos algunas clasificaciones acertadas, se encuentran muchas clasificaciones erróneas, por ejemplo en la Tabla 3:

- El perfil de *angelgdeagreda* se empareja con el de *pesadillacocina* (un perfil de *Ciberseguridad* con un perfil de *Cocina*).
- *arsitek_estudio* (perteneciente a la temática *Arquitectura*) es emparejado con *infocomputer* (perteneciente a *Ordenadores*) y con *espvideojuegos* (perteneciente a *Videojuegos*).
- El perfil de *canalcocina* se asocia con el de *icfs_uam* (un perfil de *Cocina* con uno de *Ciberseguridad*).
- *funarco* (enmarcado en la categoría *Arquitectura*) es emparejado con *edgarvasquezc* y *ccncert* (ambos perfiles de *Ciberseguridad*).
- *nintendoes* (de *Videojuegos*) es emparejado con *funarco* y con *edgarvasquezc*.

En la Tabla 4 también se encuentran clasificaciones erróneas, como las siguientes:

- *angelgdeagreda* es emparejado con *arsitek_estudio* y con *uto_dev* (perfil de *Ordenadores*).
- *arsitek_estudio* con *easysmartech* (perfil de *Ordenadores*) y con *selvaorejon* (perfil de *Ciberseguridad*).
- *canalcocina* es asociado con *capcom_es* (con temática de *Videojuegos*) y con *c1b3rwall* (con temática de *Ciberseguridad*).
- *cosasdehackers* (*Ciberseguridad*) es emparejado con *capcom_es*.
- *funarco* con *gintelseg* (*Ciberseguridad*) y con *edgarvasquezc*.
- *gintelseg* se empareja con *c1b3rwall* y *funarco*.
- *nintendoes* se asocia con *cocinatelefe* (perfil de *Cocina*) y con *funarco*.

Con estos resultados se determina que este modelo no es capaz de clasificar los perfiles por temática, ni aún siquiera variando el valor del parámetro *Topics*.

5.3 Random Projections

Este modelo está basado en el modelo TF-IDF pero con un parámetro extra que permite reducir la dimensionalidad de los vectores de salida. A continuación, se muestran los resultados obtenidos:

Perfil introducido	*Perfiles más similares*
«agdamundo»	(0.2039): «inteligencia_l» (0.1873): «edgarvasquezc»
«angelgdeagreda»	(0.2805): «sandra_vazquezb» (0.2578): «eei_icfs»
«arsitek_estudio»	(0.3797): «pesadillacocina» (0.3039): «edgarvasquezc»
«avalos_morer»	(0.4090): «pesadillacocina» (0.2945): «sandra_vazquezb»
«canalcocina»	(0.2888): «avalos_morer» (0.2302): «blissy»
«cneces»	(0.4009): «incibe» (0.3007): «easysmartech»
«cosasdehackers»	(0.2704): «karguinano» (0.2451): «uto_dev»
«funarco»	(0.2373): «uto_dev» (0.1827): «jose_strife»
«gintelseg»	(0.6131): «ivanpormor» (0.2226): «c1b3rwall»
«nintendoes»	(0.3409): «karguinano» (0.1914): «onbranding»

Tabla 5. Random Projections. Topics 50.

Para el modelo TF-IDF se observaba que los valores de similitud obtenidos rondaban el 10%. En este modelo se aprecia que los valores medios son más altos, de aproximadamente el 30% de similitud. Sin embargo, la clasificación por temática falla, encontrando emparejamientos tales como *arsitek_estudio* con *pesadillacocina*, *cosasdehackers* con *karguinano* o *nintendoes* con *karguinano*. Variar el valor del parámetro *Topics* tampoco consigue mejoras en la separación por temática.

5.4 Latent Dirichlet Allocation

Este modelo utiliza un corpus de entrenamiento basado en *Bag-of-words*, como los modelos anteriores, y un parámetro de entrenamiento *Topics*, que determina la dimensionalidad. Los resultados obtenidos son los siguientes:

Perfil introducido	*Perfiles más similares*
«agdamundo»	(0.9921): «jose_strife» (0.9654): «avalos_morer»
«angelgdeagreda»	(0.6655): «ccncert» (0.5896): «uto_dev»

«arsitek_estudio»	(0.9700): «funarco» (0.9699): «easysmartech»
«avalos_morer»	(0.9700): «agdamundo» (0.9487): «jose_strife»
«canalcocina»	(0.9999): «rosaarda» (0.9999): «karguinano»
«cneces»	(0.9979): «funarco» (0.9978): «easysmartech»
«cosasdehackers»	(0.8682): «designcarmeng» (0.7685): «ivanpormor»
«funarco»	(0.9999): «easysmartech» (0.9988): «cneces»
«gintelseg»	(0.8899): «designcarmeng» (0.8250): «ivanpormor»
«nintendoes»	(0.8707): «ign_es» (0.8322): «hardware_espana»

Tabla 6. Latent Dirichlet Allocation. Topics 50.

Perfil introducido	***Perfiles más similares***
«agdamundo»	(0.9379): «jdiaz_caneja» (0.8019): «designcarmeng»
«angelgdeagreda»	(0.2481): «etsamadrid» (0.2246): «cybercampes»
«arsitek_estudio»	(0.9976): «infocomputer» (0.7683): «easysmartech»
«avalos_morer»	(0.9762): «cosasdehackers» (0.9757): «nintendoes»
«canalcocina»	(0.9177): «karguinano» (0.3967): «cybercampes»
«cneces»	(0.4972): «pedaguitu» (0.4967): «cybercampes»
«cosasdehackers»	(0.9747): «ivanpormor» (0.9577): «pedaguitu»
«funarco»	(0.9520): «espvideojuegos» (0.5790): «nintendoes»
«gintelseg»	(0.9927): «cybercampes» (0.9924): «c1b3rwall»
«nintendoes»	(0.9551): «avalos_morer» (0.9264): «ivanpormor»

Tabla 7. Latent Dirichlet Allocation. Topics 300.

Perfil introducido	***Perfiles más similares***
«agdamundo»	(0.7582): «lascosasretro» (0.6451): «blissy»
«angelgdeagreda»	(0.3908): «uto_dev» (0.1540): «arsitek_estudio»

«arsitek_estudio»	(0.9965): «icfs_uam» (0.9916): «inteligencia_l»
«avalos_morer»	(0.9189): «easysmartech» (0.9037): «sandra_vazquezb»
«canalcocina»	(0.9094): «karguinano» (0.8485): «pesadillacocina»
«cneces»	(0.0527): «ccncert» (0.0524): «incibe»
«cosasdehackers»	(0.9136): «hardware_espana» (0.5157): «lascosasretro»
«funarco»	(0.9217): «edgarvasquezc» (0.8471): «jdiaz_caneja»
«gintelseg»	(0.9654): «ivanpormor» (0.9143): «inteligencia_l»
«nintendoes»	(0.3378): «capcom_es» (0.1331): «pedaguitu»

Tabla 8. Latent Dirichlet Allocation. Topics 1000.

A la luz de los resultados obtenidos, se observa que el modelo no es capaz de clasificar en función de la temática de los perfiles; un ejemplo claro lo tenemos en la Tabla 8, para el perfil de *angelgdeagreda*, que es asociado con los perfiles de *uto_dev* y *arsitek_estudio*, perfiles que ni siquiera entran dentro de la misma categoría.
Se observa que variar el parámetro *Topics* no mejora la clasificación del modelo.

5.5 Hierarchical Dirichlet Process

Este modelo también utiliza un corpus de entrenamiento basado en *Bag-of-words*. Los resultados obtenidos son los siguientes:

Perfil introducido	***Perfiles más similares***
«agdamundo»	(0.8496): «avalos_morer» (0.7656): «incibe»
«angelgdeagreda»	(0.5551): «agdamundo» (0.1729): «pedaguitu»
«arsitek_estudio»	(0.9999): «easysmartech» (0.9999): «nintendoes»
«avalos_morer»	(0.8690): «incibe» (0.8684): «videojuegosgame»
«canalcocina»	(0.9943): «cybercampes» (0.8951): «capcom_es»
«cneces»	(0.7708): «icfs_uam» (0.7517): «jdiaz_caneja»
«cosasdehackers»	(0.2604): «videojuegosgame» (0.2573): «lascosasretro»
«funarco»	(0.9993): «etsamadrid» (0.9979): «arsitek_estudio»
«gintelseg»	(0.7642): «ivanpormor» (0.0914): «icfs_uam»

«nintendoes»	(1.0): «easysmartech» (0.9999): «arsitek_estudio»

Tabla 9. Hierarchical Dirichlet Process.

Como en los casos anteriores, se observa que el modelo no clasifica de forma correcta, llegando a decir que un perfil es exactamente igual a uno distinto con una temática totalmente distinta a la suya propia.

5.6 Doc2Vec

Para este modelo se crea un corpus de entrenamiento formado por *Tagged Documents*. Un *Tagged Document* es una asociación entre el id asociado al documento en cuestión y la lista de todas las palabras que aparecen en el documento.

Este modelo requiere de tres parámetros de entrenamiento:

- Tamaño del vector inferido.
- Número mínimo de apariciones requeridas por término.
- Número de épocas de entrenamiento del modelo.

Se decide realizar distintas pruebas en las que se irá variando el tamaño del vector, el número mínimo de apariciones, y el número de épocas de entrenamiento.

La primera prueba se hace con un tamaño de vector de 30, número mínimo de apariciones de 5 y 10 épocas de entrenamiento.

Perfil introducido	***Perfiles más similares***
«agdamundo»	(0.9448): «angelgdeagreda» (0.9422): «pedaguitu»
«angelgdeagreda»	(0.9590): «agdamundo» (0.9187): «jdiaz_caneja»
«arsitek_estudio»	(0.7990): «etsamadrid» (0.7786): «cneces»
«avalos_morer»	(0.9046): «pedaguitu» (0.8985): «sandra_vazquezb»
«canalcocina»	(0.9114): «karguinano» (0.8953): «rosaarda»
«cneces»	(0.9110): «c1b3rwall» (0.9046): «ccncert»
«cosasdehackers»	(0.9016): «thexxlman» (0.8692): «designcarmeng»
«funarco»	(0.8635): «c1b3rwall» (0.8428): «cosasdehackers»
«gintelseg»	(0.9471): «ivanpormor» (0.9352): «c1b3rwall»
«nintendoes»	(0.8965): «espvideojuegos» (0.7718): «videojuegosgame»

Tabla 10. Doc2Vec con size 30, min_count 5, epochs 10.

Utilizando la Tabla 10, en la que se agrupaban los perfiles por temática, observamos que por regla general los perfiles más similares son aquellos que se encuentran en su misma temática, como ocurre con los perfiles de *agdamundo*, *angelgdeagreda*, *avalos_morer*, *canalcocina*, *cneces*, *gintelseg* y *nintendoes*. No sólo eso, sino que algunos perfiles son llevados por la misma persona (como es el caso de *agdamundo* y *angeldeagreda*, o de *gintelseg* e *ivanpormor*) y el sistema determina que son los más similares entre sí. En general, los resultados obtenidos son favorables.
No obstante, observamos que agrupa incorrectamente perfiles con distintas temáticas, como es el caso de *arsitek_estudio* (un perfil con temática de arquitectura) con *cneces* (un perfil de ciberseguridad). También se observa en el caso de *cosasdehackers* con *designcarmeng*, y de *funarco* con *c1b3rwall* y *cosasdehackers*.

Dado que el número de documentos no es excesivamente grande, se realizan más pruebas con el mismo tamaño de vector y el mismo número de aparición, variando el número de épocas de entrenamiento, obteniéndose los siguientes resultados:

Perfil introducido	***Perfiles más similares***
«agdamundo»	(0.7877): «pedaguitu» (0.7468): «angelgdeagreda»
«angelgdeagreda»	(0.7844): «agdamundo» (0.6440): «cneces»
«arsitek_estudio»	(0.6648): «etsamadrid» (0.6138): «designcarmeng»
«avalos_morer»	(0.6856): «cosasdehackers» (0.5727): «thexxlman»
«canalcocina»	(0.7095): «karguinano» (0.5461): «designcarmeng»
«cneces»	(0.7648): «c1b3rwall» (0.7496): «icfs_uam»
«cosasdehackers»	(0.7952): «thexxlman» (0.7044): «avalos_morer»
«funarco»	(0.6177): «designcarmeng» (0.5277): «videojuegosgame»
«gintelseg»	(0.9453): «ivanpormor» (0.5840): «c1b3rwall»
«nintendoes»	(0.5323): «espvideojuegos» (0.5178): «videojuegosgame»

Tabla 11. Doc2Vec con size 30, min_count 5, epochs 50.

Perfil introducido	***Perfiles más similares***
«agdamundo»	(0.9081): «angelgdeagreda» (0.8148): «pedaguitu»
«angelgdeagreda»	(0.8797): «agdamundo» (0.7420): «c1b3rwall»
«arsitek_estudio»	(0.6978): «agdamundo» (0.6861): «designcarmeng»

«avalos_morer»	(0.8679): «cosasdehackers» (0.7398): «cneces»
«canalcocina»	(0.8070): «karguinano» (0.7102): «incibe»
«cneces»	(0.8748): «c1b3rwall» (0.8550): «icfs_uam»
«cosasdehackers»	(0.8401): «avalos_morer» (0.7688): «thexxlman»
«funarco»	(0.8341): «designcarmeng» (0.6621): «arsitek_estudio»
«gintelseg»	(0.9719): «ivanpormor» (0.7044): «cneces»
«nintendoes»	(0.6828): «espvideojuegos» (0.6577): «videojuegosgame»

Tabla 12. Doc2Vec con size 30, min_count 5, epochs 500.

Perfil introducido	***Perfiles más similares***
«agdamundo»	(0.8724): «angelgdeagreda» (0.7134): «pedaguitu»
«angelgdeagreda»	(0.8488): «agdamundo» (0.6667): «cneces»
«arsitek_estudio»	(0.6548): «etsamadrid» (0.5638): «funarco»
«avalos_morer»	(0.7987): «cosasdehackers» (0.7157): «sandra_vazquezb»
«canalcocina»	(0.7063): «karguinano» (0.6241): «designcarmeng»
«cneces»	(0.8378): «icfs_uam» (0.7590): «pedaguitu»
«cosasdehackers»	(0.8040): «avalos_morer» (0.7478): «thexxlman»
«funarco»	(0.6775): «designcarmeng» (0.6191): «canalcocina»
«gintelseg»	(0.9677): «ivanpormor» (0.6957): «c1b3rwall»
«nintendoes»	(0.8595): «espvideojuegos» (0.6836): «capcom_es»

Tabla 13. Doc2Vec con size 30, min_count 5, epochs 2000.

A la luz de los resultados obtenidos se llega a la conclusión de que a mayor número de épocas, más aprende el modelo y consigue dar perfiles similares con mayor acierto. En el caso del modelo con las 2000 épocas, vemos que el primer resultado obtenido siempre coincide con la categoría temática del perfil elegido. El segundo resultado suele coincidir, salvo excepciones, como son el resultado para *canalcocina* (aunque hay que decir que el cuarto resultado para *canalcocina* sería *cocinatelefe*, con una similitud de 0.5746) y para *funarco* (el tercer resultado sería *etsamadrid*, con una similitud de 0.5844). Se cree que

trabajando con una base de datos de mayor volumen de documentos con más diversidad, el sistema podría obtener mejores resultados.

Habiendo determinado que, con este volumen de documentos, aumentar el número de épocas mejora la clasificación, se prueba con los parámetros de tamaño y número de aparición, para comprobar si se pueden obtener mejores resultados.

Perfil introducido	***Perfiles más similares***
«agdamundo»	(0.8034): «angelgdeagreda» (0.7914): «sandra_vazquezb»
«angelgdeagreda»	(0.8083): «agdamundo» (0.8046): «selvaorejon»
«arsitek_estudio»	(0.7961): «uto_dev» (0.7685): «selvaorejon»
«avalos_morer»	(0.8060): «selvaorejon» (0.7976): «uto_dev»
«canalcocina»	(0.8014): «karguinano» (0.7940): «agdamundo»
«cneces»	(0.7821): «selvaorejon» (0.7814): «incibe»
«cosasdehackers»	(0.8270): «thexxlman» (0.7785): «espvideojuegos»
«funarco»	(0.8015): «arsitek_estudio» (0.7817): «selvaorejon»
«gintelseg»	(0.8990): «ivanpormor» (0.8182): «selvaorejon»
«nintendoes»	(0.7992): «espvideojuegos» (0.7859): «videojuegosgame»

Tabla 14. Doc2Vec con size 100, min_count 5, epochs 2000.

Aumentar la dimensionalidad del vector de salida no solo no mejora los resultados, sino que hace que el modelo pierda la capacidad de discriminar los perfiles por temática y estilo y los mezcla entre sí, siendo más acentuado el empeoramiento cuanto más se aumenta la dimensionalidad. Por tanto, decidimos utilizar una dimensionalidad pequeña.

Por último, se comprueba si variar el número mínimo de apariciones de una palabra tiene efectos positivos en los resultados.

Perfil introducido	***Perfiles más similares***
«agdamundo»	(0.8974): «angelgdeagreda» (0.6931): «eei_icfs»
«angelgdeagreda»	(0.8874): «agdamundo» (0.6425): «eei_icfs»
«arsitek_estudio»	(0.7444): «funarco» (0.6044): «etsamadrid»
«avalos_morer»	(0.7788): «cosasdehackers» (0.7250): «thexxlman»

«canalcocina»	(0.6959): «karguinano» (0.5586): «rosaarda»
«cneces»	(0.7311): «icfs_uam» (0.6986): «sandra_vazquezb»
«cosasdehackers»	(0.8278): «thexxlman» (0.7763): «avalos_morer»
«funarco»	(0.7367): «arsitek_estudio» (0.6539): «designcarmeng»
«gintelseg»	(0.9700): «ivanpormor» (0.6419): «thexxlman»
«nintendoes»	(0.8063): «espvideojuegos» (0.6516): «capcom_es»

Tabla 15. Doc2Vec con size 30, min_count 15, epochs 2000.

En comparación con la Tabla 13 vemos que se subsanan algunos errores como la asociación errónea de *canalcocina* con *designcarmeng*, o de *funarco* con *canalcocina*. Además, ahora se consigue que cuentas que son llevadas por la misma persona (*cneces*, *icfs_uam* y *sandra_vazquezb*; *cosasdehackers*, *thexxlman* y *avalos_morer*) estén asociadas entre sí.

Por tanto, se estima que los mejores parámetros para el modelo con una base de datos pequeña son una dimensionalidad pequeña de vectores, un mínimo de aparición de palabras medio (para eliminar términos poco frecuentes y que introduzcan ruido en el perfil), y un número de épocas de entrenamiento alto.

6 Conclusiones y trabajo futuro

Durante este trabajo se analizó la posibilidad de comparar perfiles extraídos de redes sociales, buscando similitudes entre el contenido de los distintos perfiles. El objetivo final del proyecto es el de identificar qué perfiles son gestionados por la misma persona. Para ello se extrajeron cuarenta y tres perfiles con temáticas diversas, entre los que se encuentran perfiles gestionados por la misma persona, de la red social Twitter; estos perfiles fueron procesados e insertados en diferentes modelos de procesamiento del lenguaje natural, de los que se obtuvieron una serie de resultados.

6.1 Conclusiones

Tras la obtención de los resultados y el estudio de los mismos, se llegan a las siguientes conclusiones:

- Los modelos utilizados en este trabajo que discriminan textos basándose únicamente en características semánticas no dan buenos resultados para la tarea planteada y no son considerados aptos.
- El modelo Doc2Vec, que además de estudiar las características semánticas del texto también es guardar información estilística del texto, obtiene buenos resultados a la hora de clasificar los perfiles propuestos y genera listas de similitud válidas.
- Con lo expuesto en los dos puntos anteriores, se determina que es posible crear el sistema objetivo de este TFG, que clasifica perfiles similares de tal forma que es muy probable que la misma persona se halle tras varios de los perfiles seleccionados para la lista de similitud.

6.2 Trabajo futuro

Como trabajo futuro se podría tratar de mejorar el sistema de clasificación para que realizase comparaciones más robustas utilizando parámetros de naturaleza distinta a los del NPL, como podría ser la incorporación de hábitos de conexión de los perfiles a estudiar (días en los que hay más actividad y las horas de actividad del perfil), o discernir entre los distintos clientes desde los que operan con los perfiles.
También se podrían utilizar otras formas de preprocesamiento para los perfiles, antes de ser introducidos en el modelo.

Referencias

[1] https://arxiv.org/pdf/1810.04805.pdf
[2] https://arxiv.org/pdf/1903.09025v1.pdf
[3] https://openreview.net/attachment?id=rygGQyrFvH&name=original_pdf
[4] https://openreview.net/attachment?id=SJeYe0NtvH&name=original_pdf
[5] https://arxiv.org/pdf/1911.03429v1.pdf
[6] https://www.nltk.org/
[7] https://spacy.io/usage/spacy-101
[8] https://docs.allennlp.org/master/
[9] https://radimrehurek.com/gensim/about.html
[10] https://textblob.readthedocs.io/en/dev/
[11] D. Jurafsky, J. H. Martin, "Speech and Language Processing. An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition". Third Edition Draft. Pp. 65.
[12] D. Jurafsky, J. H. Martin, "Speech and Language Processing. An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition". Third Edition Draft. Pp. 112-114.
[13] D. Jurafsky, J. H. Martin, "Speech and Language Processing. An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition". Third Edition Draft. Pp. 128-129.
[14] Bradford, R., An Empirical Study of Required Dimensionality for Large-scale Latent Semantic Indexing Applications, Proceedings of the 17th ACM Conference on Information and Knowledge Management, Napa Valley, California, USA, 2008, pp. 153–162.
[15] Landauer, Thomas K., and Dumais, Susan T., "Latent Semantic Analysis", Scholarpedia, 3(11):4356, 2008.
[16] http://users.ics.aalto.fi/ella/publications/randproj_kdd.pdf
[17] Blei, D. M., Ng, A. Y., and Jordan, M. I. (2003). "Latent Dirichlet allocation". Journal of Machine Learning Research, 3(5), 993–1022.
[18] http://proceedings.mlr.press/v15/wang11a/wang11a.pdf
[19] https://radimrehurek.com/gensim/auto_examples/tutorials/run_doc2vec_lee.html#sphx-glr-auto-examples-tutorials-run-doc2vec-lee-py
[20] https://radimrehurek.com/gensim/auto_examples/tutorials/run_word2vec.html#sphx-glr-auto-examples-tutorials-run-word2vec-py
[21] https://cs.stanford.edu/~quocle/paragraph_vector.pdf

Glosario

NLP	Procesamiento del Lenguaje Natural
BERT	Bidirectional Encoder Representations from Transformers
ERASER	Evaluating Rationales And Simple English Reasoning
NLTK	Natural Language Toolkit
API	Application Programing Interface
TF-IDF	Term Frequency – Inverse Document Frequency
LSI	Latent Semantic Indexing
SVD	Singular Value Decomposition
LDA	Latent Dirichlet Allocation
pLSA	Probabilistic Latent Semantic Analysis
LSA	Latent Semantic Analysis
HDP	Hierarchical Dirichlet Process